

Build System

Build System Description

2012/10/26

Version 1.3

**The content of this document is highly confidential
and should be handled accordingly.**

Confidential

These coded instructions, statements, and computer programs contain proprietary information of Nintendo and/or its licensed developers and are protected by national and international copyright laws. They may not be disclosed to third parties or copied or duplicated in any form, in whole or in part, without the prior written consent of Nintendo.

Table of Contents

1	Introduction	5
2	Quick Start	6
2.1	Preparing Development Tools	6
2.1.1	Build Environment	6
2.2	Building NintendoWare Libraries	6
2.2.1	Installing NintendoWare	6
2.2.2	Environment Variables Needed for Builds.....	6
2.2.3	Building the NintendoWare Tree	7
3	Build System	8
3.1	Coding the Makefile	8
3.1.1	Example Makefile for an Application Build	8
3.1.2	Makefile for a User Library Build	9
3.1.3	Using Precompiled Headers.....	9
3.2	Build Versions	10
3.2.1	Debug Version	10
3.2.2	Develop Version	11
3.2.3	Release Version	11
3.3	make Targets	11
3.3.1	clean and clobber Targets	11
3.4	Build Switches	11
3.5	Variables Used With a Makefile	12
3.6	Macros Passed to Compiler or Assembler	14
3.7	buildtools Directory	14
3.7.1	commondefs.mk	15
3.7.2	modulerules.mk	16
	Revision History	17

Code

Code 3-1	Example Makefile for Application Build	8
Code 3-2	Example Makefile for Library Build	9
Code 3-3	Example C++ Source File (precompiled.cpp) That Includes Header to Be Precompiled.....	9
Code 3-4	Example Header File (precompiled.h) to Be Precompiled.....	9
Code 3-5	Example (Partial) C++ Source File (main.cpp) That Uses Precompiled Header.....	10
Code 3-6	Example (Partial) Makefile That Uses Precompiled Header	10

Tables

Table 2-1 Environment Variables That Need to Be Set.....	7
Table 3-1 NintendoWare Files with Code for Common Procedures	8
Table 3-2 Targets for <code>make</code>	11
Table 3-3 Build Switches	12
Table 3-4 Variables That Can Be Used With the Makefile	12
Table 3-5 Macros Passed to the Compiler or Assembler	14
Table 3-6 Subdirectories of the <code>buildtools</code> Directory.....	14
Table 3-7 Platform Settings File	15
Table 3-8 Compiler Settings File	15
Table 3-9 Compiler Rules File	16
Table 3-10 Emulator Rules File	16

1 Introduction

This document explains how to build the NintendoWare for Cafe (NintendoWare) libraries and sample programs. It also explains the NintendoWare Build System.

Note: This document does not describe the PC runtime development environment that supports building and executing on a PC.

2 Quick Start

2.1 Preparing Development Tools

Current versions of the NintendoWare libraries support builds for Cafe-SDK. When using the NintendoWare libraries, the environment must be configured for Cafe-SDK.

2.1.1 Build Environment

Current NintendoWare builds have been validated on Microsoft Windows 7. The SDK and tools required to build and debug the NintendoWare libraries and any applications that use the libraries are.

- Green Hills Software MULTI
- Cygwin
- Cafe-SDK
- CAT-DEV v2 or CAT-DEV v3

2.2 Building NintendoWare Libraries

The procedure for building NintendoWare libraries and sample programs is described below.

2.2.1 Installing NintendoWare

NintendoWare is provided as a ZIP file. Extract the contents to an appropriate location. Once the ZIP file is extracted, a directory named `NW_Cafe` is created on the local disk. All NintendoWare for Cafe files can be found in this directory.

2.2.2 Environment Variables Needed for Builds

The environmental variables needed for builds are set with the Cygwin shell environment started by executing `NW_Cafe/shell.bat`. However, environmental variables such as `CAFE_ROOT` must be set separately in order to use `shell.bat`. See *NintendoWare for Cafe Runtime Library Startup Guide* for details on `shell.bat`.

A different method to set the environmental variables needed for builds is to use the Cygwin shell started up when executing the Cafe-SDK `cafe_sdk/cafe.bat`. When using `cafe.bat`, the `NW4F_ROOT` environment variable required for builds is not set so it must be set separately.

The following shows the environmental variables needed when using `NW_Cafe/shell.bat` or `cafe_sdk/cafe.bat`.

Table 2-1 Environment Variables That Need to Be Set

Environment Variable	Description	Example
NW4F_ROOT	Specifies the NintendoWare install path as an absolute path. When <code>shell.bat</code> is used, it is set internally.	<code>NW4F_ROOT = C:\NW_Cafe</code>
CAFE_ROOT	Specifies the Cafe-SDK install path as an absolute path. When <code>cafe.bat</code> is used, it is set internally.	<code>CAFE_ROOT = C:\cafe_sdk</code>
GHS_ROOT	Specifies the Green Hills Software MULTI install path as an absolute path.	<code>GHS_ROOT = C:\ghs\multi534</code>
CYGWIN_PATH	Specifies the Cygwin install path as an absolute path.	<code>CYGWIN_PATH = C:\cygwin</code>

Below, the NintendoWare installation directory is notated as `$NW4F_ROOT`.

2.2.3 Building the NintendoWare Tree

Once `NW4F_ROOT` has been set, you can build libraries and sample programs by running Cygwin to execute `make`, which is located inside the `Library` subdirectory of the NintendoWare root directory `$NW4R_ROOT`.

To build the libraries and sample programs, enter:

```
cd $NW4F_ROOT/Library
make
```

If nothing is specified, then the development version of the libraries and sample programs is built. To build the debug version of the libraries and sample programs, enter:

```
cd $NW4F_ROOT/Library
make NW_BUILD=Debug
```

3 Build System

3.1 Coding the Makefile

A set of files has been prepared in NintendoWare to simplify the coding of `Makefile` for applications that use NintendoWare libraries. These files contain code for commonly used procedures.

Table 3-1 NintendoWare Files with Code for Common Procedures

Content	File
Directory	<code>\$NW4F_ROOT/Library/build/make</code>
Definition file for variables and more	<code>commondefs.mk</code>
Definition file for compile procedures	<code>modulerules.mk</code>

Include these two files in `Makefile` to create an application that uses NintendoWare libraries. The `commondefs.mk` file contains the build environment settings, and the `modulerules.mk` file contains the build method.

3.1.1 Example Makefile for an Application Build

Code 3-1 shows an example `Makefile` for an application build. To study an actual `Makefile`, look at one used to compile a library's sample program or another program.

Code 3-1 Example Makefile for Application Build

```
#! make -f
#-----
SUBDIRS =

#-----
SRCS      = main.cpp title.cpp game.cpp
TARGET_BIN = main.elf

include $(NW4F_ROOT)/Library/build/make/commondefs.mk

#-----
do-build: $(TARGETS)

include $(NW_BUILDTOOLSDIR)/modulerules.mk
```

- To execute a `Makefile` that is in a subdirectory, code the name of that directory in `SUBDIRS`.
- `SRCS` specifies the source file.
- `TARGET_BIN` specifies the execution file.
- `NW_BUILDTOOLSDIR` is set with `commondefs.mk` to indicate `$NW4F_ROOT/Library/build/make`. In this example, this variable is used when

`modulerules.mk` is included, but the same result is possible by including the
`$(NW4F_ROOT)/Library/build/make` path, the same as `commondefs.mk`.

3.1.2 Makefile for a User Library Build

Code 3-2 shows an example `Makefile` for a local library created by the user.

Code 3-2 Example Makefile for Library Build

```
#!/ make -f
#-----
SUBDIRS =

#-----
SRCS      = utility.cpp math.cpp
TARGET_LIB = libgame.a

include $(NW4F_ROOT)/Library/build/make/commondefs.mk

INSTALL_TARGETS = $(TARGETS)
INSTALL_DIR     = $(PROJECT_ROOT)/lib

#-----
do-build: $(TARGETS)

include $(NW_BUILDTOOLS_DIR)/modulerules.mk
```

- `SRCS` specifies the source file.
- `TARGET_LIB` specifies the library file.
- `INSTALL_TARGETS` specifies any file that should be installed.
- `INSTALL_DIR` specifies the directory where you want `INSTALL_TARGETS` to install the specified files.

3.1.3 Using Precompiled Headers

A precompiled header feature has been added to Green Hills Software MULTI 5.3.15. To use the precompiled header feature in the NintendoWare Build System, create a C++ file that includes the header file to be precompiled and specify that file in the `PRECOMPILED_CPP` variable.

In the following example, the C++ file that includes the header file to be precompiled is named `precompiled.cpp`, and the header file to be precompiled is named `precompiled.h`.

Code 3-3 Example C++ Source File (precompiled.cpp) That Includes Header to Be Precompiled

```
#include "precompiled.h"
```

Code 3-4 Example Header File (precompiled.h) to Be Precompiled

```
#include <cafe.h>
#include <nw/math.h>
#include <nw/ut.h>
```

Code 3-5 Example (Partial) C++ Source File (main.cpp) That Uses Precompiled Header

```
#include "precompiled.h"

...

int main(int argc, char **argv)
{
    ...
}
```

Code 3-6 Example (Partial) Makefile That Uses Precompiled Header

```
...
SRCS      = main.cpp sub.cpp

# CPP source that includes header file to be precompiled
PRECOMPILED_CPP = precompiled.cpp

include $(NW4F_ROOT)/Library/build/make/commondefs.mk
...
```

3.2 Build Versions

NintendoWare Build System can create three different build versions: Debug, Develop, and Release. Which version is created depends on the build switch that is set when the `make` command is executed. If no build switch is set, the Release version is built by default.

NW_BUILD Value	Description
Debug, DEBUG, debug	Builds the Debug version.
Develop, DEVELOP, develop	Builds the Develop version.
Release, RELEASE, release	Builds the Release version.

For example, execute the following to perform a debug build.

```
make NW_BUILD=Debug
```

3.2.1 Debug Version

This version links NintendoWare libraries with the Debug version of the SDK library. Because the Debug version is assumed to be used with the debugger, optimization is not performed during compilation, and inline functions are not expanded.

For the Debug version of NintendoWare libraries, `ASSERT` is used to check the validity of function input arguments.

3.2.2 Develop Version

This version links the Release version of the SDK library with the Develop version of the NintendoWare libraries (the Develop version libraries do not exist in the SDK). The Develop version performs optimization during compilation and automatically expands the inline functions.

Even for the Develop version of the NintendoWare libraries, `ASSERT` is used to check the validity of function input arguments.

3.2.3 Release Version

This version links NintendoWare libraries with the SDK library's Release version. For the Release version, the options are the same as the Develop version. The Release version of NintendoWare libraries omits the check for validity of function input arguments with `ASSERT`.

3.3 make Targets

A number of `make` targets can be used with NintendoWare Build System. Table 3-2 lists these targets.

Table 3-2 Targets for `make`

Command	Action
<code>make build</code>	Starts compilation and creates final target.
<code>make install</code>	Installs (copies) files created by <code>make build</code> into another directory.
<code>make full</code>	Creates files for all build versions.
<code>make full-install</code>	Installs (copies) all version files to another directory.
<code>make clean</code>	Deletes version-specific files created by <code>make build</code> .
<code>make clobber</code>	Completely deletes all files created by <code>make build</code> .

3.3.1 clean and clobber Targets

The `clean` target deletes intermediate and execution files that were created during the build process of a selected build version (Debug, Develop, or Release).

The `clobber` target deletes all files created during builds, regardless of the build version. Installed libraries and tool executables are also deleted.

3.4 Build Switches

Table 3-3 shows the build switches that can be used with NintendoWare when executing a `Makefile`.

Table 3-3 Build Switches

Switch	Description
NW_BUILD	Specifies target build version. The default is Develop. - Debug(, DEBUG, debug): Performs Debug version build. - Develop(, DEVELOP, develop): Performs Develop version build. - Release(, RELEASE, release): Performs Release version build.
NW_PLATFORM	Specifies target platform. Currently, only CAFE can be specified and it is the default. CAFE: Build using Cafe-SDK.
NW_CCTYPE	Specifies which compiler to use. Currently, only GHS can be specified and it is the default. GHS: Set when Green Hills Software MULTI is used for the compiler.
NW_EMTYPE	Specifies which debugger to use. The debugger is automatically specified based on the setting for NW_PLATFORM. Currently, only GATDEV can be specified.
NW_USELIBS	Sets the include paths, library paths, and library files for libraries in the specified library package. If this switch is not specified, settings are made for all libraries in all of the installed library packages. By default, this switch is not specified.
NW_DEBUGINFO	In the Release version (NW_BUILD is Release), normal debugging information is disabled. To enable debugging information in the Release version, define this variable.
NW_NODEBUGINFO	In the Debug version (NW_BUILD is Debug) and Develop version (NW_BUILD is Develop), normal debugging information is enabled. To disable debugging information in the Debug and Develop versions, define this variable.

3.5 Variables Used With a Makefile

Table 3-4 shows the variables that can be used with a `Makefile` for NintendoWare.

Table 3-4 Variables That Can Be Used With the Makefile

Variable Name	Description
SUBDIRS	Lists directories for the <code>make</code> processes you want to link. Initial value: None.
SUBMAKES	Lists the <code>Makefile</code> for the <code>make</code> processes you want to link. Initial value: None.

Variable Name	Description								
SRCS	Lists the source files that you want to compile and assemble. The <code>make</code> process tries to make object files for these listed files using the compiler or assembler for the programming language indicated by file extension. The table below shows the relationship between file extensions and languages. <table> <tr> <th>Extension</th><th>Programming Language</th></tr> <tr> <td><code>.c</code></td><td>C compiler</td></tr> <tr> <td><code>.cpp</code></td><td>C++ compiler</td></tr> <tr> <td><code>.s</code></td><td>PowerPC assembler</td></tr> </table>	Extension	Programming Language	<code>.c</code>	C compiler	<code>.cpp</code>	C++ compiler	<code>.s</code>	PowerPC assembler
Extension	Programming Language								
<code>.c</code>	C compiler								
<code>.cpp</code>	C++ compiler								
<code>.s</code>	PowerPC assembler								
EXT_OBJS	Set to link an object that has already been compiled externally.								
SRCDIR	Specifies the directory of the source files. Initial value: <code>./src</code> .								
INCDIR	Specifies the directory where a private include file (that is, an include file specific to a module) is located. Optional. Initial value: <code>./include</code> .								
OBJDIR	Depending on the <code>make</code> process, the object files are output to these directories.								
DEPDIR	Depending on the <code>make</code> process, the dependent (depend) files are output to these directories.								
BINDIR	Depending on the <code>make</code> process, the executable files are output to these directories.								
LIBDIR	Depending on the <code>make</code> process, the library files are output to these directories.								
LCFILE	Specifies the linker command file (<code>.lcf</code>) used with the build. Initial value: <code>\$(SDK_INCDIR)/revolution/eppc.\$(ARCH_TARGET).lcf</code>.								
TARGET_BIN	Specifies target files that are executable on the console (<code>.elf</code> format). Target files are made by linking object files of the compiled/assembled source files specified by <code>SRCS</code> .								
TARGET_LIB	The object files of the compiled/assembled source files specified by <code>SRCS</code> are archived to form a library. Specify this when you want to create a library file. Note that you cannot set <code>TARGET_BIN</code> and <code>TARGET_LIB</code> at the same time.								
TARGET_OBJ	Specify this when you want to create an object file as a target. Generally, this is not used.								
TARGETS	Items with paths relative to <code>TARGET_BIN</code> and <code>TARGET_LIB</code> are set automatically. The directory in which the <code>.elf</code> and <code>.a</code> files are stored changes depending on the build target. This variable has been prepared to handle this change.								
INSTALL_TARGETS	Lists files to be installed. Typically, the <code>TARGETS</code> variable is specified for this setting. Initial value: None.								

Variable Name	Description
INSTALL_DIR	Describes the destination where the files specified by <code>INSTALL_TARGETS</code> will be installed. Initial value: None.
LINCLUDES	Specifies the directories for any other include files.
LLIBRARY_DIRS LLIBRARIES	If there are other libraries, list their directories in <code>LLIBRARY_DIRS</code> and their files in <code>LLIBRARIES</code> .
LDIRT_CLEAN LDIRT_CLOBBER	Specifies object and other files to be deleted when the <code>make clean</code> or <code>make clobber</code> command is executed. When <code>make clean</code> is executed, the files specified in <code>LDIRT_CLEAN</code> are deleted. When <code>make clobber</code> is executed, all files specified in both <code>LDIRT_CLEAN</code> and <code>LDIRT_CLOBBER</code> are deleted.

3.6 Macros Passed to Compiler or Assembler

NintendoWare Build System passes a number of macros to the compiler so that the source contents can be changed according to build conditions. Table 3-5 lists the macros that get passed to the compiler or assembler.

Table 3-5 Macros Passed to the Compiler or Assembler

Macro	When Macro Is Defined
NW_DEBUG	Defined for a Debug build.
NW_DEVELOP	Defined for a Develop build.
NW_RELEASE	Defined for a Release build.
NW_GHS	Defined when using Green Hills Software MULTI as the compiler.
NW_CAFE	Defined when a build is being made using Cafe-SDK.

3.7 buildtools Directory

The `build/make` subdirectory of the `Library` directory holds the files needed to build the applications that use NintendoWare libraries and NintendoWare. You can configure the settings required for using NintendoWare by performing an `include` on the `commondefs.mk` and `modulerules.mk` files from inside this subdirectory in the `Makefile`.

Table 3-6 shows the subdirectories that are in the `make` directory. These subdirectories hold the files that are included in (or used by) the `commondefs.mk` and `modulerules.mk` files.

Table 3-6 Subdirectories of the buildtools Directory

Directory Name	Directory Contents
platform	Files for configuring the settings for each platform.

Directory Name	Directory Contents
compiler	Files for configuring the settings for each compiler.
emulator	Files for configuring the settings for each console emulator used to execute a program.
script	Source of the script used for builds.
shader	Files for building the shader.

3.7.1 commondefs.mk

The `commondefs.mk` file defines the main variables needed to build NintendoWare libraries. The following main settings are configured with this file.

- Destination directory where each library will be installed
- Include paths
- Library paths
- Library files passed to the linker

The `commondefs.mk` file contains a number of build-related files. The following is a simple description of the included files.

```
$NW4F_ROOT/Library/build/make/platform/commondefs.pftype.<platform_name>.mk
```

This file describes platform-related settings. By changing the value of `NW_PLATFORM`, you can switch the file to be included. Table 3-7 describes the available settings file.

Table 3-7 Platform Settings File

Variable	Value	File Read	Description
NW_PLATFORM	CAFE	commondefs.pftype.CAFE.mk	Settings file for Cafe-SDK.

```
$NW4F_ROOT/Library/build/make/compiler/commondefs.cctype.$(NW_CCTYPE).mk
```

This file describes compiler-related settings. By changing the value of `NW_CCTYPE`, you can switch the file to be included. Table 3-8 describes the available settings file.

Table 3-8 Compiler Settings File

Variable	Value	File Read	Description
NW_CCTYPE	GHS	commondefs.cctype.GHS.mk	Settings file for Green Hills Software MULTI.

```
$NW4F_ROOT/Library/<library name>/build/<library name>/commondefs.mk
```

The directory for each library holds a `commondefs.mk` file that provides information related to the libraries included in that library package. The `commondefs.mk` file contains the library include paths, the library paths, and the library files included in the library.

3.7.2 modulerules.mk

The `modulerules.mk` file describes rules for target build. It includes the settings file for each compiler. Following is a simple description of the files included in `modulerules.mk`.

```
$NW4F_ROOT/Library/build/make/compiler/modulerules.cctype.$(NW_CCTYPE).mk
```

This file describes the target rules for each compiler. By changing the value of `NW_CCTYPE` you can switch the file to be included. Table 3-9 describes the available rule file.

Table 3-9 Compiler Rules File

Variable	Value	File Read	Description
NW_CCTYPE	GHS	modulerules.cctype.GHS.mk	Rules for Green Hills Software MULTI.

```
$NW4F_ROOT/Library/build/make/emulator/modulerules.emtype.$(NW_EMTYPE).mk
```

This file describes the settings for each debugger. By changing the value of `NW_EMTYPE`, you can switch the file that will be included. Table 3-10 describes the available rule file.

Table 3-10 Emulator Rules File

Variable	Value	File Read	Description
NW_EMTYPE	CATDEV	modulerules.emtype.CATDEV.mk	Rules for CAT-DEV.

Revision History

Version	Revision Date	Category	Description
1.3	2012/10/26	Added	3.1.3 Using Precompiled Headers Added description of using precompiled headers.
1.2	2012/05/10	Deleted	3.4 Build Switches Deleted the description of the <code>NW_ASSERT_OFF</code> switch, which was not functioning.
1.1	2012/03/28	Added	3.4 Build Switches Added <code>NW_DEBUGINFO</code> and <code>NW_NODEBUGINFO</code> .
1.0	2011/09/05	—	Initial version.

All company and product names in this document are the trademarks or registered trademarks of their respective companies.

© 2011–2013 Nintendo

The contents of this document cannot be duplicated, copied, reprinted, transferred, distributed, or loaned in whole or in part without the prior approval of Nintendo.